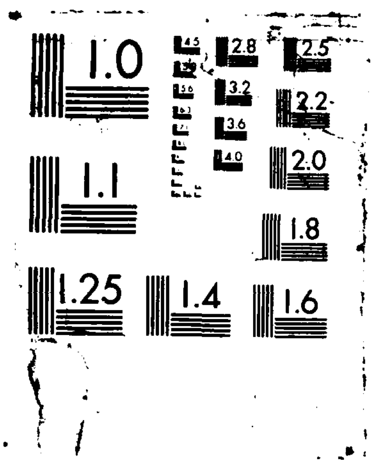
AD-A190 884 CONCURRENCY EFFICIENCY USER'S MANUAL(U) ILLINOIS UNIV AT URBANA CENTER FOR SUPERCOMPUTING RESEARCH AND DEVELOPMENT A D MALONY 30 OCT 87 CSRD-675 1/1

UNCLASSIFIED AFOSR-TR-87-1985 F49620-86-C-0136 F/G 12/5 NL


1.0
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6

DTIC FILE COPY (2)

AD-A190 884

# REPORT DOCUMENTATION PAGE

ELECTE JAN 2 5 1988 S D

1b. RESTRICTIVE MARKINGS

2a. SECURITY CLASSIFICATION AUTHORITY

2b. DECLASSIFICATION / DOWNGRADING SCHEDULE

3. DISTRIBUTION / AVAILABILITY OF REPORT
Approved for public release; distribution unlimited.

4. PERFORMING ORGANIZATION REPORT NUMBER(S)
CSRD Report No. 675

5. MONITORING ORGANIZATION REPORT NUMBER(S)
AFOSR-TR- 87-1985

6a. NAME OF PERFORMING ORGANIZATION
The Board of Trustees of the University of Illinois

6b. OFFICE SYMBOL (If applicable)

7a. NAME OF MONITORING ORGANIZATION
AFOSR/NM

6c. ADDRESS (City, State, and ZIP Code)
506 S. Wright St.
Urbana, IL 61801

7b. ADDRESS (City, State, and ZIP Code)
AFOSR/NM
Bldg 410
Bolling AFB DC 20332-6448

8a. NAME OF FUNDING/SPONSORING ORGANIZATION
AFOSR

8b. OFFICE SYMBOL (If applicable)
NM

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
~~AFOSR~~ F49620-86-C-0136

8c. ADDRESS (City, State, and ZIP Code)
AFOSR/NM
Bldg 410
Bolling AFB DC 20332-6448

10. SOURCE OF FUNDING NUMBERS

| PROGRAM ELEMENT NO. | PROJECT NO. | TASK NO. | WORK UNIT ACCESSION NO. |
|---|---|---|---|
| 61102F | 2304 | A3 | |

11. TITLE (Include Security Classification)
Concurrency Efficiency User's Manual

12. PERSONAL AUTHOR(S)
Allen D. Malony

13a. TYPE OF REPORT
Internal Report

13b. TIME COVERED
FROM 10/1/86 TO 9/30/87

14. DATE OF REPORT (Year, Month, Day)
Oct. 30, 1987

15. PAGE COUNT

16. SUPPLEMENTARY NOTATION

17. COSATI CODES

| FIELD | GROUP | SUB-GROUP |
|---|---|---|
| | | |
| | | |

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This document explains how to use a facility for measuring concurrency efficiency on the Alliant FX/8 implemented as part of CSRD's modifications to the Concentrix operating system. A brief overview of the concurrency efficiency analysis is presented first. The CSRD implementation is then described to point out the measurement's limitations. Concurrency efficiency measurements are directly controlled by the user program. Instructions for determining CEFF values from within a program are given. Concurrency efficiency statistics for the entire program are often desired. A tool for generating this data without requiring user program modification is described. Finally, we give some suggestions on the use of CEFF results in association with other program performance information.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT
☐ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS

21. ABSTRACT SECURITY CLASSIFICATION

22a. NAME OF RESPONSIBLE INDIVIDUAL
Maj. John P. Thomas

22b. TELEPHONE (Include Area Code)
(202) 767-5026

22c. OFFICE SYMBOL
NM

DD FORM 1473, 84 MAR

83 APR edition may be used until exhausted.
All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

# Concurrency Efficiency User's Manual

*Allen D. Malony*
CSRD

AFOSR-TR- 87-1985

## 1. Introduction

Knowing the time spent by a parallel program in various modes of concurrent execution is important for characterizing program performance. A basic performance metric derived from measuring the amount of time $i$ processors are active, where $i$=1,$n$ and $n$ is the total number of processors available, is the *concurrency efficiency*, or *CEFF*. Essentially, *CEFF* shows the multiprocessor utilization obtained by the program. It is an interesting performance metric because the inverse represents a bound on the maximum speedup possible for the measured program run. However, concurrency efficiency results should be interpreted in relation to other measurements such as the actual speedup obtained.

This document explains how to use a facility for measuring concurrency efficiency on the Alliant FX/8 implemented as part of CSRD's modifications to the Concentrix™ operating system. A brief overview of the concurrency efficiency analysis is presented first. The CSRD implementation is then described to point out the measurement's limitations. Concurrency efficiency measurements are *directly controlled by the user program*. Instructions for determining *CEFF* values from within a program are given. Concurrency efficiency statistics for the entire program are often desired. A tool for generating this data without requiring user program modification is described. Finally, we give some suggestions on the use of *CEFF* results in association with other program performance information.

## 2. CEFF Analysis

If $T_i$ is the amount of time a program spends executing with $i$ processors active, where $i$=1,$n$ and $n$ is the total number of processors, *CEFF* is defined as:

$$CEFF = \left( \sum_{i=1}^{i=n} i * T_i \; / \; n \right) * T * 100\%$$

$$where \quad T = \sum_{i=1}^{i=n} T_i$$

Given the concurrency timing information, $T_i$, it is simple to derive *concurrency utilization*

results, $CU_i$, as the percentage of time $i$ processors are active:

$$CU_i = \frac{T_i}{T} * 100\%$$

By definition, a processor is active when it is is executing user code. However, there is no differentiation in the measurement of the type of user code being executed. The *CEFF* metric indicates the average percentage of the processors used by the program and the *CU* values give a breakdown of execution time spent in each concurrent execution state[1]. The inverse of *CEFF* gives the upper bound on program speedup possible for this run. It is an upper bound because portions the user code executed may not contribute to the overall program progress; such is the case with synchronization operations. Whereas a low *CEFF* value implies a low level of concurrent processor activity and, therefore, a poor speedup situation, a high *CEFF* value is only an indication of high processor concurrency and does not necessarily reflect good parallel performance. As will be seen later, *CEFF* and *CU* values must be considered with other performance metrics to determine the degree of ***effective parallelism*** being achieved.

## 3. CEFF Implementation

Ideally, changes in the number of active processors should be detected to measure the time spent in the different concurrent states. However, this measurement procedure is undesirable for the FX/8 since it could severely restrict parallel program performance. Instead, an approach based on the same sampling mechanism used for profiling is implemented.

When concurrency efficiency measurements are enabled, the program is interrupted every 10 msec and the state of the processor complex is sampled. The number of active processors is easily determined by comparing each processor's program counter to a known *idle* PC. A counter associated with each concurrent state is kept for the program; initially, the counters are set to zero. For each sample, the concurrency counter associated with the current number of active processors is incremented. The time spent in concurrent state $i$, $T_i$, is calculated by multiplying the $i$th concurrency count value, $N_i$, by 10 msec. The *CEFF* and *CU* values can then be easily computed as shown above.

Because of the 10 msec sampling procedure, the concurrent state timing data is only a statistical approximation to the actual concurrency timing information. The primary assumption is that the concurrent state indicated by the sample is the same throughout the entire sampling period. Obviously, for a single sample period, this assumption is invalid. However, the negative effects of the assumption lessen as the number of samples increase. Only when the total concurrency measurement time is significantly greater than the sampling period can statistical accuracy be achieved. It is important to remember the statistical accuracy when interpreting the *CEFF* results.

[1] A concurrent state is defined for each possible number of active processors. Concurrent state $i$ is the state where only $i$ processors are active.

## 4. User–Controlled CEFF Measurements

Because of the need to periodically interrupt program execution, concurrency efficiency measurements pay a high system overhead. Thus, concurrency measurements are made only when enabled by the program. This section describes how a program enables concurrency measurement, accesses the measured concurrency data, and computes the *CEFF* statistics.

### 4.1. Enabling CEFF Measurement

The system function *configceff()* in the *csrd* library should be used for enabling and disabling *CEFF* measurement. The C description is given below:

```
configceff(flag)
int   flag;          /* 1 - enable, 0 - disable */
```

*configceff(1)* enables the measurements and *configceff(0)* disables the measurements; a negative value is returned if already enabled or disabled, respectively. *CEFF* measurements are disabled upon program entry.

One benefit of user–controlled concurrency measurements is the ability to determine *CEFF* results for different parts of the program. Each section of the program to be measured should be bracketed by *configceff(1)* and *configceff(0)* as shown below:

```
<read CEFF measurement data>
configceff(1);
<program section>
configceff(0);
<read CEFF measurement data and compute results>
```

Reading the concurrency measurement data and computing the results are discussed below.

### 4.2. CEFF Variables

As described above, the operating system maintains *CEFF* measurement data using *n* concurrency counters; one for each concurrent state – eight for the FX/8. Actually, there are two sets of concurrency counters which exist as part of a larger structure containing additional process measurement information. One set contains the counters for the current process. The second set of concurrency counts are the summed *CEFF* measurements for child processes[2]. The process measurement data structures, which include the *CEFF* measurement variables, are declared in <sys/csrdetc.h>. The declarations of interest for this document are reproduced below:

```
struct ceff
{   u_long   cpuutil[8];      /* concurrency counters */
}
```

[2] If the program does not fork any child processes, the children concurrency counts will alway be zero.

```
struct csrdetc
{   ...                              /* other measurement fields     */
    u_char       csrd_ceffon;        /* CEFF measurement enable flag */
    struct ceff csrd_ceff;           /* current process CEFF counts  */
    struct ceff csrd_cceff;          /* children CEFF counts         */
}

extern struct csrdetc csrdetc;
```

The *csrdetc* data structure has been implemented as part of the user program's read–only virtual address space. This allows convenient access to the *CEFF* measurement data from within the program. However, the write–protection requires *configceff()* to be a system call because it must set and reset the *csrd_ceffon* flag. *csrd_ceffon* can be referenced directly for testing purposes.

It should be noted that children concurrency counts are not updated until a child process completes. Also, each child process will get its own copy of the *csrdetc* data structure. The *csrd_ceffon* variable is set to whatever it was in the parent process. However, the *csrd_ceff* and *csrd_cceff* concurrency counts are initialized to zero.

### 4.3. Accessing CEFF Measurement Data

Accessing *CEFF* measurement data is very simple. Three global symbols have been defined in the *csrd* library that are accessible in C programs using the following declarations:

```
external struct ceff ceff;    /* &ceff  =&(csrdetc.csrd_ceff)    */
external struct ceff cceff;   /* &cceff =&(csrdetc.csrd_cceff)   */
external u_char      ceffon;  /* &ceffon=&(csrdetc.csrd_ceffon)  */
```

Notice from the comments that the external names are defined to have the same address as their corresponding *csrdetc* fields. Using these external names, the concurrency counters can be accessed directly. The following operation copies the children concurrency counters to a local *ceff* variable, *ceff_local*:

```
struct ceff     ceff_local;

ceff_local = cceff;
```

A similar operation will copy the current process concurrency counts. Although the concurrency counters can be accessed individually, the procedure of copying the concurrency counts to a local buffer should be followed when the user wants the counter values to be consistent in time. Remember the counts are dynamically changing during program execution and computing *CEFF* statistics will require time consistent concurrency counts.

### 4.4. Computing CEFF Statistics

Returning to the procedure for performing *CEFF* measurement on a section of the program, there are two places where the concurrency information should be sampled: immediately before

enabling *CEFF* measurement and immediately after disabling. This reason for this is clear. The concurrency counts show the total time accumulated in the concurrent states for the current process and its children. To determine the concurrency efficiency for a particular program section, the difference in the two concurrency count samples must be computed.

Assuming we have a set of concurrency count values representing either a program section or the accumulated counts for the program, there are various values that can be computed. If $N_i$ represents the concurrency count for $i$ processors active, $T_i = N_i * 10\ msec$. *CEFF* and *CU* values are computed from the above formulas. Knowing *CEFF* and the number of processors, *numprocs* where $numprocs \leq n$, the average number of processors active, *CAVG*, can also be computed as $CAVG = CEFF * numprocs$.

Because of the simplicity of these concurrency calculations, the user is left to implement the *CEFF* statistics of interest[3]. Obviously, the user will decide whether to use the current process concurrency counts, the children process concurrency counts or both for computing *CEFF* statistics.

### 4.5. Program Compilation

To perform *CEFF* measurements, the user program must include the following files:

```
sys/param.h
sys/resource.h
sys/csrdetc.h
machine/vmparam.h
machine/mpcpiadr.h
sys/mplock.h
```

The user program must then be compiled with the *csrd* library.

## 5. Automatic Program CEFF Measurements

In many cases, the user will want to determine *CEFF* results for the program as a whole. A program, *ceff*, has been written to run the user's program and print out concurrency efficiency statistics. The *ceff* command format is:

```
ceff <user program> <user program arguments>
```

The results produced by *ceff* include $T_i$, $CU_i$, $T$, *CAVG*, and *CEFF*. An example of the output is shown below:

```
CONCURRENCY EFFICIENCY RESULTS

# active CEs          seconds          concurrency %
```

[3] A program could be provided that reads concurrency count samples from a file and computes accumulated and difference *CEFF* statistics.

| | | |
|---|---|---|
| 1 | 3.37 | 28.25% |
| 2 | 0.39 | 3.27% |
| 3 | 0.30 | 2.51% |
| 4 | 0.51 | 4.28% |
| 5 | 0.75 | 6.29% |
| 6 | 1.02 | 8.55% |
| 7 | 1.49 | 2.49% |
| 8 | 4.10 | 4.37% |

```
total seconds          = 11.93
average concurrency    =  5.05
concurrency efficiency = 63.07%
```

A manual page is available for the *ceff* program on the CSRD Alliant machines.

## 6. Interpreting CEFF Results

It is important to remember the concurrency efficiency results only represent measurements of processor activity. No analysis is made of what the processors are actually doing when they are active. Thus, the *CEFF* results should not necessarily be interpreted as effective parallelism. It is true, however, that concurrency efficiency does establish an upper bound on speedup.

Suppose whenever the program is executing concurrently, all active processors are executing independently. In this case, concurrency efficiency will reflect effective parallelism because it is assumed each processor is doing real work. However, if dependencies exist between processors during concurrent execution, the processors will appear active even when they are performing synchronization operations or waiting for dependencies to be satisfied. Because such activity represents overhead and does not contribute to real work, the concurrency efficiency will indicate a parallelism higher than what is effectively being achieved.

*CEFF* results can be used with other measurements to better characterize program performance. For instance, speedups from 1 processor to $n$ processors can help to clarify effective parallelism. Suppose a program achieves a speedup $S$=6 going from 1 to 8 processors and a *CEFF* value of 80% ($CAVG$ = 6.4). Although only 80% of the processors are utilized on average, almost all of the 6.4 average processor concurrency is being used effectively. In this case, the user might conclude that physical parallelism, i.e. keeping more processors active, is the problem. However, $S$=2 for a program with *CEFF*=80% indicates a low effective parallelism, likely due to synchronization overhead or a large sequential component.

The *CU* measurements are interesting because they give a histogram of concurrent activity. The $CU_i$ values where $i<n$ are important because they represent periods of reduced parallelism when processors are actually idle. $CU_1$ is most important since it is the percentage of time the program is executing sequentially. The $CU_1$ value can be plugged directly into Amdahl's equation to get the projected maximum program speedup for $p$ processors[4]. For the results produced by *CEFF* above:

[4] We are using Amdahl's equation $\lim_{p\to\infty} S_p = 1 / (1-F_p)$ where $F_p$ is the fraction of time all $p$ processors are active. We assume that the percentages of all concurrent activity are summed to get $F_p$. Thus, the calculated asymptotic speedup is actually optimistic.

$$\lim_{p \to \infty} S_p = \frac{1}{CU_1 \,/\, 100\%} = \frac{1}{.2825} = 3.54$$

Although $CAVG$=5.05, the asymptotic speedup is limited by the significant sequential component.

## 7. CEFF and Fortran

All of the discussion above has been directed towards C programs. Slight modifications are necessary for Fortran programs. The *configceff()* system call is used exactly as before. Likewise, there are no differences in the *CEFF* statistics computations between C and Fortran once the data has been retrieved. Only the accessing of the concurrency counts is different.

All Fortran programs must do to reference the concurrency counts is correctly declare the external names described above. The following does this and should be included in Fortran programs:

```
byte       ceffon
integer    ceff(8), cceff(8)

common     /ceff/   ceff
common     /cceff/  cceff
common     /ceffon/ ceffon
```

As before, the external names address the current process and children process concurrency counters, and the enable flag. Fortran can index the concurrency counter arrays directly to access individual counters. However, the copying of the entire counter array to a local buffer is still recommended.

END
DATE
FILMED
5-88
DTIC